Intro to Qt on Android

Niels Mayer,
nielsmayer.com
niels.mayer@gmail.com

Why Qt 5.7?

Per BogDan Vatra's "All About Qt on Android" presentation at QtCon16:

- Android splash screen improvements
- Helper functions to run C++ code on the Android UI thread
- Android services
- Qt Quick Controls 2 with "Material style - a style based on the Google Material Design Guidelines."
- Qt Purchasing: API to support in-app purchasing with Google Play.

Why Qt 5.8?

More complete Qt Quick Controls 2 Module

- Added new QML types: Dialog, DialogButtonBox, MenuSeparator, RoundButton, and ToolSeparator.
- Added ripple/hover effects, and System theme, to the Material style.
- Added new experimental (QtLabs) platform QML types that work on Android: MenuBar, Menu, MessageDialog.

Qt 5.8+ and Vulkan, NVIDIA CodeWorks and Shield

Building the latest greatest for Android AArch64 (with Vulkan teaser)
https://blog.qt.io/blog/2017/02/24/building-latest-greatest-android-aarch64-vulkan-teaser/

‘The basic Vulkan enablers, including VulkanWindow are currently scheduled for Qt 5.10, with support for Windows, Linux/X11, and Android. (the list may grow later on)’

Qt 5.8 “Refactored the Qt Quick scene graph to remove its OpenGL dependency, enabling backends based on other graphics APIs such as Vulkan or Direct3D.”

Setup the development environment for Android

Prerequisites per ‘Getting started with Qt for Android’:

- The Android SDK Tools
- The Android NDK
- Java SE Development Kit (JDK) v6 or later. You can also use OpenJDK on Linux.
- On Windows, you need the following additional installations:
    - Apache Ant v1.8 or later
    - Android Debug Bridge (ADB) driver on the Windows platform to enable USB debugging. The default USB driver on Windows does not allow debugging. For details about how to get the USB driver, see http://developer.android.com/sdk/win-usb.html.

Caveat: Linux preferred development environment

BogDan Vatra - ‘GNU/Linux is currently the recommended platform, yielding best development experience’

I have no experience or advice for Android/Qt development on Mac or Windows

Issues regarding usb debugging on windows, requires special driver/etc

Issues with 32 bit ‘adb’ requiring i386 libs installed on Linux

Maybe sidestep these issues using NVIDIA CodeWorks for Android?

Simpler to install ‘NVIDIA CodeWorks for Android’ ?

http://docs.nvidia.com/gameworks/index.html#developertools/mobile/codeworks_android/codeworks_android_1r6.htm

Contains everything needed for Android development in one package/download:

Android SDK r24.4.1_u1. Android Build Tools r24.0.1, Android Platform Tools r24.0.1, Android Support Library r23.2.1, Android Support Repository Library r35, Android APIs, Android NDK64 r12b, Google USB Driver r11, JDK 1.8.0_77, Eclipse 4.4, CDT 8.2.0, ADT 24.0.2, Apache Ant 1.8.2 Gradle 2.9

Features; Automatic updates of above; Android-TV development, Vulkan, etc.

Platforms: Windows 8.1 (64-bit), Windows 7 (64-bit), Mac OS X 10.9, Ubuntu Linux x64 (v14.04).

(YMMV: perhaps CodeWorks for Android solves issues setting up Android development environment for Windows or Mac?)

Installation Tips

Qt SDK 5.7 or 5.8 to get latest Android developments

Android SDK version 22+

Android NDK r10e is recommended. (BogDan claims ndk r11+ has gcc bug that produces buggy debug builds).

Use Gcc and not Clang...

Android Clang issue means continue using GCC

Use gcc and not clang -- per BogDan “clang creates 40% larger binaries”

Clang now the recommended Android native compiler, with gcc ‘deprecated”

NDK: Gcc 4.9 is the last version maintained and 5.0 will not be supported.

Per Bug report “GCC in the android NDK is now deprecated”, Android-clang is already supported in Qt 5.7.1 and 5.8, and will be available in LTS 5.6.2. However, “clang produces way too big libs, check https://github.com/android-ndk/ndk/issues/133”

Setup

- Install ndk and sdk, e.g. using NVIDIA CodeWorks for Android installer.
- Put Android device into ‘developer mode’

  Settings->About <device> -> Build number (tap 6 times for Android 6, 7 times for Android 7)

- Enable usb debugging:

  Settings->General->Developer Options->Debugging->USB Debugging->On

- On Linux be sure you have USB Permissions set (may be default already).

  See https://developer.android.com/studio/run/device.html for details.

The Application’s “Android Manifest”

Setup Android manifest for app per ‘Publishing to Google Play’

Select Create Templates under Build settings to create the template files such as *AndroidManifest.xml* and other resources.

Note: Qt Quick Controls with native Android style require API 11 (Android v3.0) or later.

- Add Application name and Application icon.
- Permissions list has all the required permissions.
- Features list has the software or hardware features that your application depends on. For example, NFC.
- Setup Android Services (Qt 5.7+) to run background processes without UI.

Use QtCreator to create Android Manifest

Projects->Build&Run->Android->Build->Build Android APK->Create Templates

- Creates Android manifest file for project
  - Set build sdk version.
  - Setup package signing key etc.
  - Can use Ministro II service to install Qt libs once and share across apps
  - Or setup a shared location so that Qt Libs and components needn't be packaged w/ each app.
  - Default is to bundle the needed Qt libs and components with app. Slow. Large.
  - Bundled is probably how one wants to deliver apps on Play Store, despite size (or not)
    - Avoid dependency on Ministro where Qt version updates could break your app.
    - But… "***User reviews***: +46MB for just a damn app drawer?! Wow, this is literally junk. More then 46MB and all this launcher is, is a white button with an app drawer. No widgets, literally no settings or customization. You should go work for Apple."

Example AndroidManifest.xml for Mobile,Tablet & TV

```
<manifest package="com.nielsmayer.mandelbrot" …

 <application android:name="org.qtproject.qt5.android.bindings.QtApplication"
              android:allowBackup="true"
              android:hardwareAccelerated="true"
              android:label="-- %%INSERT_APP_NAME%% --" >
...
<activity android:name="org.qtproject.qt5.android.bindings.QtActivity"
android:configChanges="orientation|uiMode|screenLayout|screenSize|smallestScreenSize|layoutDirection|locale|fontScale|
keyboard|keyboardHidden|navigation"
                    android:screenOrientation="landscape"
                    android:launchMode="singleTop"
                    android:label="-- %%INSERT_APP_NAME%% --" >
          <intent-filter>
             <action android:name="android.intent.action.MAIN"/>
             <category android:name="android.intent.category.LEANBACK_LAUNCHER" /> ...
```

Example App: qt quick controls2 Material Design

Gallery App: .../Qt5.8.0/Examples/Qt-5.8/quickcontrols2/gallery/gallery.pro

Unfortunately this is a lousy mobile app demo. More a desktop/tablet app with Material-looking design elements and a “Drawer Menu” that is swipeable. Each menu item allows interaction with Material-styled UI components.

- Must go to Options Menu, set Material style and restart.
- Is it really Material design? Won’t really give you a ‘material’ app on its own.
- At least Text fields now can be selected and pasted, but not cut/copied on Android via long-touch, (but not with ‘Universal’ style on Desktop?)
- No bindings for Android-TV. No “back button” handling -- exits app!

Example App: Ekke's QtQuick controls2 Demo

Drawer_nav_x: drawer_nav_x/drawer_nav_x.pro

- Requires Qt 5.7+ to use qtquick controls2
- Author claims app runs on iOS too, giving Material-design-looking apps ala Gmail and other google apps on iphone? (Is it precedent to make this acceptable for iOS Qt apps?)
- Option to show 'shortcut actions on bottom-bar in portrait mode
- Overall better example template for a complete 'Material' app. Still no Android TV compatibility.
- Claims to handle 'back button' but it is flaky and broken.

Example App: Ben Lau's 'Quick Android' Library

Quickandroid: quickandroid/examples/quickandroidexample/quickandroidexample.pro

- Does not use QtQuick controls2, or controls1, and therefore not requiring Qt 5.7 or 5.8 (requires 5.5+, will work with LTS 5.6 release and those needing to stick to GPLv2 compliance; under nonrestrictive Apache-style license).
- Set of QtQuick 2.0 UI components implemented Google's Material Design: "Back" key navigation; Auto-scale according to system's DP value; page transition animation.
- Drawable Image provider: Load image resource from Android resource style file tree,Tint image at load time, Choose the best image according to current resolution automatically.
- Message queue between C++/Qt and Java/Android code - Auto conversion between C++ and Java data type. No need to write in JNI.
- Supports better integration with Android, e.g. using native image picker, camera, notifications, bottom drawer, back-button does the right thing.
- Text field/editor actually works! Handles cut/copy/paste/selection/etc in Android-compliant fashion.

See also:

- androidnative.pri (W.I.P. componentized update to quickandroid)
- qmlpromise (deferred and asynchronous computation as QML declarative object)
- quickios (QML Theme and Component Library for iOS)
- Qtino.SharingKit (sharing framework for Android)

For more info …

- Qt5.8 Android Support Top level
- Creating Android Services
- Including third-party Android libraries in an application
- Deploying to the Device
- Publishing to Google Play
- Platform Notes
- Video Intro: QtWS16: All about Qt on Android: Say hello to Qt on Android
- Video deep dive: QtCon16: All about Qt on Android - Practical Qt on Android JNI (pdf slides). See also An introduction to JNI on Android, the Qt way, Using Android Studio to extend and debug the Java parts of Qt App, access and use Android Java API from your Qt on Android using JNI.
- Video deep dive: QtWS15- Mastering Qt Android (pdf slides from QtDD14)